Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Read and Write Data

Working with data: text formats

ASCII or text file formats

Advantages of working with text formats:

- They are usually human-readable.
- They tend to be simple structures.
- It is relatively easy to write code to interpret them.

Disadvantages include:

- Inefficient storage for big data volumes.
- Most people invent their own format so there is a lack of standardisation.

Using python to read text formats

As we have seen Python has a great toolkit for reading files and working with strings.

In this example we use a file that we found on the web, and then adapt some code to read it into a useful, re-usable form.

Our example file

We found a suitable data set on the web:

https://www.metoffice.gov.uk/climate/uk/summaries/datasets

Met Office monthly weather statistics for the UK since 1910.

Header

```
1  UK Rainfall (mm)
2  Areal series, starting from 1910
3  Allowances have been made for topographic, coastal and urban effects
4  Seasons: Winter=Dec-Feb, Spring=Mar-May, Summer=June-Aug, Autumn=Sep
5  Values are ranked and displayed to 1 dp. Where values are equal, ran
6  Data are provisional from December 2014 & Winter 2015. Last updated
7
```

Lines numbers (for reference only)

Year	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP
1910	111.4	126.1	49.9	95.3	71.8	70.2	97.1	140.2	27.0
1911	59.2	99.7	62.1	69.0	52.2	77.0	43.3	69.3	69.4
1912	111.7	79.5	128.2	36.1	58.2	124.5	92.3	167.6	57.1
1913	1	1	1	31.2	102.9	81.5	63.8	33.7	44.5	73.7
1914				24.3	52.3	59.6	52.5	94.4	80.1	57.2
1	1	1		65.0	65.9	53.2	37.7	124.7	81.6	54.9
1				88.3	70.2	93.0	71.4	73.3	92.0	54.8
1917				74.2	63.6	69.2	73.5	61.0	166.7	76.0
1918	1			42.9	49.6	65.0	42.4	120.4	84.4	182.2
1919	120.1	59.8	118.4	80.9	29.8	64.9	50.1	90.3	82.4
1920	139.7	96.1	109.6	106.1	94.4	58.7	123.5	70.4	78.9
1921	149.3	24.1	103.9	38.9	62.8	17.7	59.6	109.7	53.5
1922	124.1	114.7	72.4	86.1	63.4	52.4	117.6	99.5	79.9
1923	104.5	152.6	51.1	79.1	78.4	35.7	91.5	128.5	112.6
1924	103.9	39.0	37.3	75.7	121.6	71.9	122.6	113.0	133.3

UK Rainfall (mm)
Areal series, starting from 1910
Allowances have been made for topographic, coastal and urban effects
Seasons: Winter=Dec-Feb, Spring=Mar-May, Summer=June-Aug, Autumn=Sep
Values are ranked and displayed to 1 dp. Where values are equal, ran
Data are provisional from December 2014 & Winter 2015. Last updated

Data (first 9 columns)

Year	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP
1910	111.4	126.1	49.9	95.3	71.8	70.2	97.1	140.2	27.0
1911	59.2	99.7	62.1	69.0	52.2	77.0	43.3	69.3	69.4
1912	111.7	79.5	128.2	36.1	58.2	124.5	92.3	167.6	57.1
1913	123.4	57.1	131.2	102.9	81.5	63.8	33.7	44.5	73.7
1914	78.8	114.9	124.3	52.3	59.6	52.5	94.4	80.1	57.2
1915	118.7	141.1	55.0	65.9	53.2	37.7	124.7	81.6	54.9
1916	108.9	140.3	88.3	70.2	93.0	71.4	73.3	92.0	54.8
1917	73.6	33.8	74.2	63.6	69.2	73.5	61.0	166.7	76.0
1918	105.2	106.3	42.9	49.6	65.0	42.4	120.4	84.4	182.2
1919	120.1	59.8	118.4	80.9	29.8	64.9	50.1	90.3	82.4
1920	139.7	96.1	109.6	106.1	94.4	58.7	123.5	70.4	78.9
1921	149.3	24.1	103.9	38.9	62.8	17.7	59.6	109.7	53.5
1922	121.1	111.7	72.1	86.1	69.1	52.1	117.6	99.5	79.9
1923	104.5	152.6	51.1	79.1	78.4	35.7	91.5	128.5	112.6
1924	103.9	39.0	37.3	75.7	121.6	71.9	122.6	113.0	133.3

ban effects where relationships are found to exist.
Autumn=Sept-Nov. (Winter: Year refers to Jan/Feb).
equal, rankings are based in order of year descending.
st updated 07/04/2015

Data (last 8 columns)

AUG	SEP	OCT	NOV	DEC	WIN	SPR	SUM	AUT	ANN
0.2	27.0	89.4	128.4	142.2	---	217.0	307.5	244.8	1148.9
9.3	69.4	91.5	141.3	188.4	301.0	183.4	189.6	302.2	1022.4
7.6	57.1	116.2	106.9	163.7	379.6	222.5	384.4	280.3	1242.0
4.5	73.7	103.0	125.9	86.6	344.2	315.6	142.1	302.6	1027.4
0.1	57.2	61.8	139.3	203.3	280.3	236.3	227.0	258.3	1118.6
1.6	54.9	74.5	84.7	183.8	463.1	174.0	244.0	214.0	1075.6
2.0	54.8	173.5	129.6	107.1	433.0	251.5	236.7	358.0	1202.6
6.7	76.0	168.0	121.4	61.5	214.4	207.0	301.2	365.4	1042.4
4.4	182.2	106.6	97.6	134.4	272.9	157.5	247.1	386.4	1136.9
0.3	82.4	70.4	100.9	166.5	314.3	229.2	205.2	253.7	1034.5
0.4	78.9	75.7	82.2	104.6	402.3	310.0	252.6	236.8	1139.8
9.7	53.5	82.5	72.7	130.6	278.1	205.7	187.0	208.7	905.4
9.5	79.9	16.9	66.5	127.3	369.1	221.9	269.5	193.1	1050.8
8.5	112.6	157.8	112.8	115.5	384.4	208.6	255.6	383.1	1220.0
3.0	133.3	122.8	83.2	153.6	258.3	234.6	307.4	339.2	1177.7

ban effects where relationships are found to exist.
Autumn=Sept-Nov. (Winter: Year refers to Jan/Feb).
equal, rankings are based in order of year descending.
st updated 07/04/2015

Look! A missing value!

AUG	SEP	OCT	NOV	DEC	WIN	SPR	SUM	AUT	ANN
0.2	27.0	89.4	128.4	142.2	---	217.0	307.5	244.8	1148.9
9.3	69.4	91.5	141.3	188.4	301.0	183.4	189.6	302.2	1022.4
7.6	57.1	116.2	106.9	163.7	379.6	222.5	384.4	280.3	1242.0
4.5	73.7	103.0	125.9	86.6	344.2	315.6	142.1	302.6	1027.4
0.1	57.2	61.8	139.3	203.3	280.3	236.3	227.0	258.3	1118.6
1.6	54.9	74.5	84.7	183.8	463.1	174.0	244.0	214.0	1075.6
2.0	54.8	173.5	129.6	107.1	433.0	251.5	236.7	358.0	1202.6
6.7	76.0	168.0	121.4	61.5	214.4	207.0	301.2	365.4	1042.4
4.4	182.2	106.6	97.6	134.4	272.9	157.5	247.1	386.4	1136.9
0.3	82.4	70.4	100.9	166.5	314.3	229.2	205.2	253.7	1034.5
0.4	78.9	75.7	82.2	104.6	402.3	310.0	252.6	236.8	1139.8
9.7	53.5	82.5	72.7	130.6	278.1	205.7	187.0	208.7	905.4
9.5	79.9	46.9	66.5	127.3	369.4	221.9	269.5	193.4	1050.8
8.5	112.6	157.8	112.8	115.5	384.4	208.6	255.6	383.1	1220.0
3.0	133.3	122.8	83.2	153.6	258.3	234.6	307.4	339.2	1177.7

Let's write some code to read it

We'll need:

- To read the header and data separately
- To think about the data structure (so it is easy to retrieve the data in a useful manner).

Let's put into practice what we have learnt:

- Use NumPy to store the arrays
- But we'll need to test for missing values and use Masked Array (numpy.ma)

Example code (and data)

Please refer to the example code:

```
example_code/test_read_rainfall.py
```

And data file:

```
example_data/uk_rainfall.txt
```

Reading the header

UK Rainfall (mm)
Areal series, starting from 1910
Allowances have been made for topographic, coastal and urban effects where relationships are found to exist.
Seasons: Winter=Dec-Feb, Spring=Mar-May, Summer=June-Aug, Autumn=Sept-Nov. (Winter: Year refers to Jan/Feb).
Values are ranked and displayed to 1 dp. Where values are equal, rankings are based in order of year descending.
Data are provisional from December 2014 & Winter 2015. Last updated 07/04/2015

Reading the header

```
UK Rainfall (mm)
Areal series, starting from
Allowances have been made fo
      and urban effects whe
      found to exist.
Seasons: Winter=Dec-Feb, Sp
      Summer=June-Aug, Autu
      Year refers to Jan/Fe
Values are ranked and displ
      values are equal, ran
      order of year descend
Data are provisional from De
      2015. Last updated 07
```

Line 1 is important information.

Other lines are useful information.

Let's capture the metadata in:

- location: UK
- variable: Rainfall
- units: mm

Reading the header

```
def readHeader(fname):
    # Open the file and read the relevant lines
    with open(fname) as f:
        head = f.readlines()[:6]

    # Get important stuff
    location, variable, units = head[0].split()
    units = units.replace("(", "").replace(")", "")

    # Put others lines in comments
    comments = head[1:6]
    return (location, variable, units, comments)
```

Test the reader

```
>>> (location, variable, units, comments) = \
        readHeader("../example_data/uk_rainfall.txt")

>>> print(location, variable, units)
UK Rainfall mm

>>> print(comments[1])
Allowances have been made for topographic, coastal and
urban effects where relationships are found to exist.
```

Write a function to handle missing data properly

```
import numpy.ma as MA


def checkValue(value):
    # Check if value should be a float
    # or flagged as missing
    if value == "---":
        value = MA.masked
    else:
        value = float(value)
    return value
```

Reading the data (part 1)

```
import numpy.ma as MA

def readData(fname):
    # Open file and read column names and data block
    with open(fname) as f:

        # Ignore header
        for i in range(7):
            f.readline()

        col_names = f.readline().split()
        data_block = f.readlines()

    # Create a data dictionary, containing
    # a list of values for each variable
    data = {}
```

UK Rainfall (mm)
Areal series, starting from 1910
Allowances have been made for topographic, coastal and urban effects
Seasons: Winter=Dec-Feb, Spring=Mar-May, Summer=June-Aug, Autumn=Sep
Values are ranked and displayed to 1 dp. Where values are equal, ran
Data are provisional from December 2014 & Winter 2015. Last updated

Data (first 9 columns)

Year	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP
1910	111.4	126.1	49.9	95.3	71.8	70.2	97.1	140.2	27.0
1911	59.2	99.7	62.1	69.0	52.2	77.0	43.3	69.3	69.4
1912	111.7	79.5	128.2	36.1	58.2	124.5	92.3	167.6	57.1
1913	123.4	57.1	131.2	102.9	81.5	63.8	33.7	44.5	73.7
1914	78.8	114.9	124.3	52.3	59.6	52.5	94.4	80.1	57.2
1915	118.7	141.1	55.0	65.9	53.2	37.7	124.7	81.6	54.9
1916	108.9	140.3	88.3	70.2	93.0	71.4	73.3	92.0	54.8
1917	73.6	33.8	74.2	63.6	69.2	73.5	61.0	166.7	76.0
1918	105.2	106.3	42.9	49.6	65.0	42.4	120.4	84.4	182.2
1919	120.1	59.8	118.4	80.9	29.8	64.9	50.1	90.3	82.4
1920	139.7	96.1	109.6	106.1	94.4	58.7	123.5	70.4	78.9
1921	149.3	24.1	103.9	38.9	62.8	17.7	59.6	109.7	53.5
1922	121.1	111.7	72.1	86.1	69.1	52.1	117.6	99.5	79.9
1923	104.5	152.6	51.1	79.1	78.4	35.7	91.5	128.5	112.6
1924	103.9	39.0	37.3	75.7	121.6	71.9	122.6	113.0	133.3

Reading the data (part 2)

```
# Add an entry to the dictionary for each column
for col_name in col_names:

    data[col_name] = MA.zeros(len(data_block), 'f',
                              fill_value = -999.999)
```

Reading the data (part 3)

```
# Loop through each value: append to each column
for (line_count, line) in enumerate(data_block):
    items = line.split()

    for (col_count, col_name) in enumerate(col_names):
        value = items[col_count]
        data[col_name][line_count] = checkValue(value)

return data
```

Testing the code

```
>>> data = readData("../example_data/uk_rainfall.txt")
>>> print(data["Year"])
[ 1910.  1911.  1912. ...

>>> print(data["JAN"])
[ 111.40000153   59.20000076  111.69999695 ...

>>> winter = data["WIN"]
>>> print(MA.is_masked(winter[0]))
True
>>> print(MA.is_masked(winter[1]))
False
```

ban effects where relationships are found to exist.
Autumn=Sept-Nov. (Winter: Year refers to Jan/Feb).
equal, rankings are based in order of year descending.
st updated 07/04/2015

Look! A missing value!

AUG	SEP	OCT	NOV	DEC	WIN	SPR	SUM	AUT	ANN
0.2	27.0	89.4	128.4	142.2	---	217.0	307.5	244.8	1148.9
9.3	69.4	91.5	141.3	188.4	301.0	183.4	189.6	302.2	1022.4
7.6	57.1	116.2	106.9	163.7	379.6	222.5	384.4	280.3	1242.0
4.5	73.7	103.0	125.9	86.6	344.2	315.6	142.1	302.6	1027.4
0.1	57.2	61.8	139.3	203.3	280.3	236.3	227.0	258.3	1118.6
1.6	54.9	74.5	84.7	183.8	463.1	174.0	244.0	214.0	1075.6
2.0	54.8	173.5	129.6	107.1	433.0	251.5	236.7	358.0	1202.6
6.7	76.0	168.0	121.4	61.5	214.4	207.0	301.2	365.4	1042.4
4.4	182.2	106.6	97.6	134.4	272.9	157.5	247.1	386.4	1136.9
0.3	82.4	70.4	100.9	166.5	314.3	229.2	205.2	253.7	1034.5
0.4	78.9	75.7	82.2	104.6	402.3	310.0	252.6	236.8	1139.8
9.7	53.5	82.5	72.7	130.6	278.1	205.7	187.0	208.7	905.4
9.5	79.9	46.9	66.5	127.3	369.4	221.9	269.5	193.4	1050.8
8.5	112.6	157.8	112.8	115.5	384.4	208.6	255.6	383.1	1220.0
3.0	133.3	122.8	83.2	153.6	258.3	234.6	307.4	339.2	1177.7

What about CSV or tab-delimited?

The above example will work exactly the same with a tab-delimited file (because the string split method splits on white space) .

If the file used commas (CSV) to separate columns then you could use:

```
line.split(",")
```

Or try the Python "csv" module

There is a python "csv" module that is able to read text files with various delimiters. E.g.:

```
>>> import csv
>>> r = csv.reader(open("../example_data/ uk_rainfall.txt "))
>>> for row in r:
...     print(row)

['Date', 'Time', 'Temp', 'Rainfall']
['2014-01-01', '00:00', '2.34', '4.45']
['2014-01-01', '12:00', '6.70', '8.34']
['2014-01-02', '00:00', '-1.34', '10.25']
```

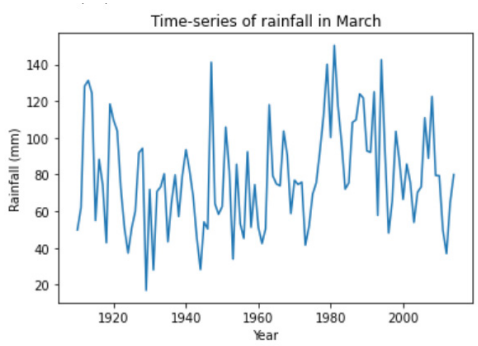
See: https://docs.python.org/3.7/library/csv.html

But really, use “pandas” instead!

```
# Read the first 6 lines, as before

fpath = "example_data/uk_rainfall.txt"
metadata = [line.strip() for line in
                          open(fpath).readlines()[:6]]

for item in metadata:
    print("Item:", item)
```

Why pandas?

```
import pandas as pd
df = pd.read_csv(fpath, skiprows=6, sep="\s+",
      index_col=0, na_values="---")

# DataFrames have loads of methods
df.columns.tolist()
df.index.name
df.describe()
df.info()
df.loc[1910:1915, "WIN"]
df.MAR.plot(title="Time-series of rainfall in March",
            ylabel="Rainfall (mm)")
```



https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html